- Expressions and Operators (Chapter 4).
- Statements (Chapter 5).
- Functions (Chapter 6).

Very brief and concise!

Skip!!

Expression

- An expression is composed of one or more operands that are combined by operators.
- To understand expressions involved more than one operator, it is necessary to understand **precedence** and **associativity**.
- Precedence determines how operators are grouped in a compound expression.
- Associativity determines how operators at the same precedence level are grouped.
- Table 4.4 (see note).

- Arithmetic operators are left-associative (group left to right when the precedence level is the same).

```
5+10*20/2;
```
⇨

- Logical operators: && and || have a lower precedence level

```
string s("Expressions in C++ are composed...");
auto it = s.begin();
while (it != s.end() && !isspace(*it)) {
   *it = toupper(*it); ++it;
}
```

Be the compiler and explain the behavior!

- The IO operators are left associative:

```
cout << "hi" << " there" << endl;
```

⇨

- Assignment operator is right associative and has a low precedence.

```
i = j = 0;
```
⇨

```
int i;
while ((i = get_value()) != 42) {
   // do something ...
}
```

Do we need the parentheses marked in red?

- Be brevity: experienced C++ programmers value being concise.

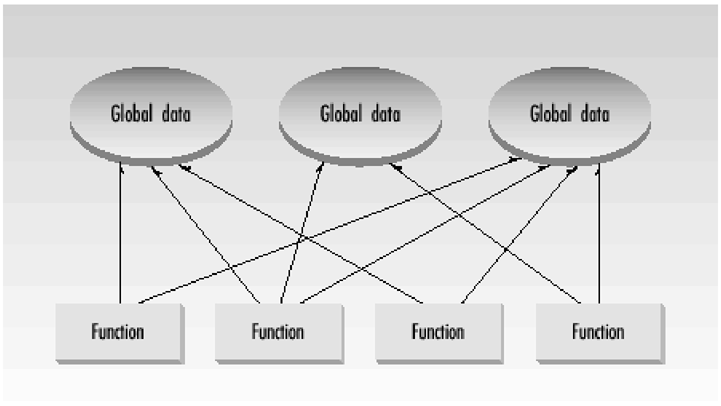
```
cout << *iter << endl;
iter++;
```

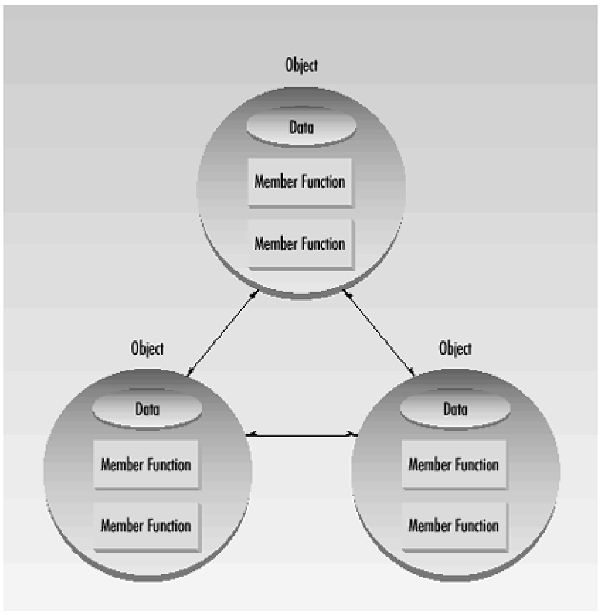
⇨ `cout << *iter++ << endl;`

- *iter++ is commonly used in C++ and we are comfortable if we know the facts:
  - Postfix operator ++ has a higher precedence level than dereference operator *.
  - Postfix ++ returns a copy of its original, unincremented operand (pp. 149)

Functions

- A function can be thought of as a programmer-defined operations.
- Functions play a key role in procedural programming and an important role in object-oriented programming.





```
// return the greatest common divisor
int gcd(int v1, int v2)
{   while (v2) {
        int temp = v2;
        v2 = v1 % v2;
        v1 = temp;
    }
    return v1;
}
```

- A function is uniquely defined by
  - its name
  - its operand types (parameters).
- The actions of function are specified in a block, referred to as the function body.
- Every function has an associated return type.

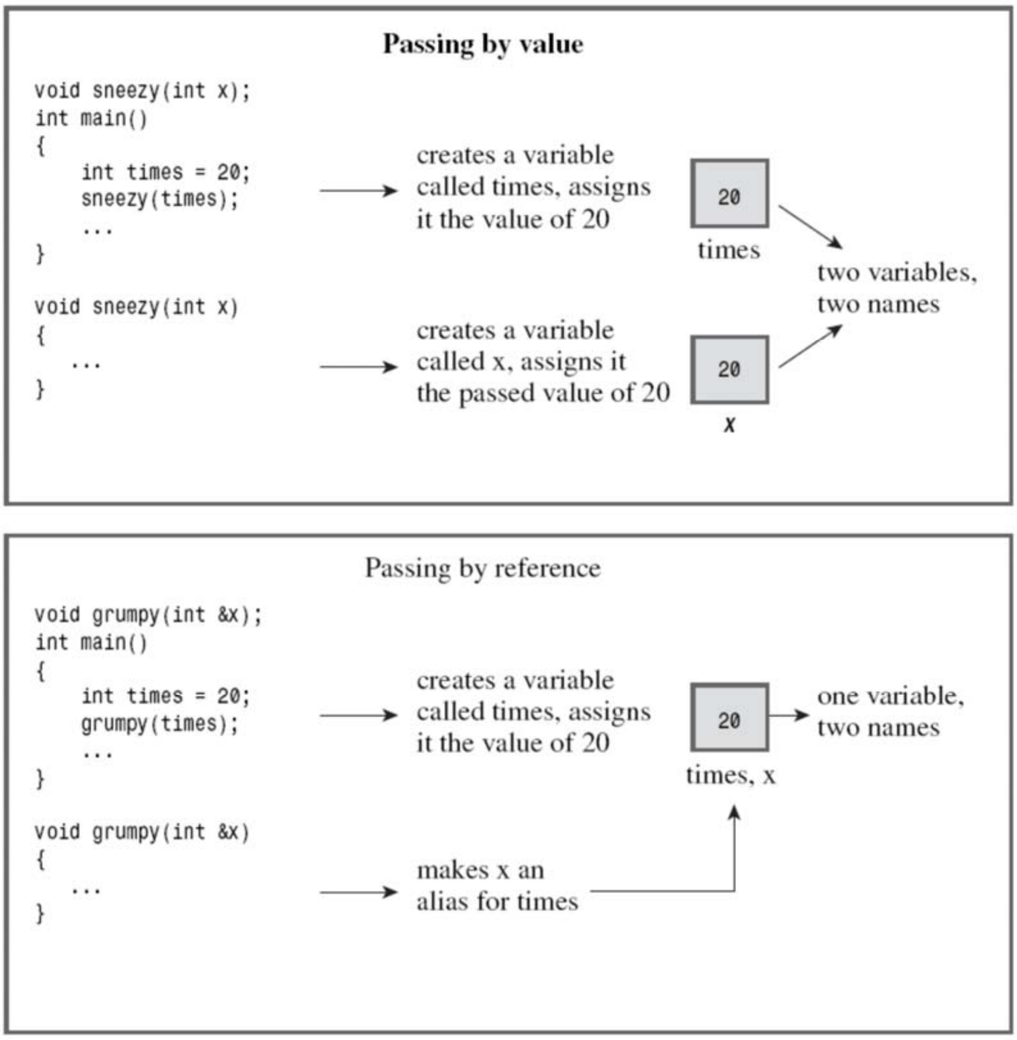
```
// get values from standard input
cout << "Enter two values: \n";
int i, j;
cin >> i >> j;
// call gcd on arguments i and j
// and print their greatest common divisor
cout << "gcd: " << gcd(i, j) << endl;
```

- We use call operator (a pair of parentheses) to invoke a function.

Functions: Argument Passing

- Parameters and passing arguments
  - Pass nonreference and reference parameters.
  - Pass const reference parameters.
  - Pass pointer and array

Pass nonreference and reference parameters


Passing by value

```
void sneezy(int x);
int main()
{
    int times = 20;
    sneezy(times);
    ...
}

void sneezy(int x)
{
   ...
}
```

creates a variable called times, assigns it the value of 20
20
times
two variables, two names
creates a variable called x, assigns it the passed value of 20
20
x

Passing by reference

```
void grumpy(int &x);
int main()
{
    int times = 20;
    grumpy(times);
    ...
}

void grumpy(int &x)
{
   ...
}
```

creates a variable called times, assigns it the value of 20
20
times, x
one variable, two names
makes x an alias for times

- We also use reference parameters when passing a large object to a function to avoid copy. For example, objects of most class types or large arrays.
- When the only reason to make a parameter a reference is to avoid copying the argument, the parameter should be const reference. (why?)

```
// compare the length of two strings
// avoid copies of strings because it could be long
bool isShorter(const string &s1, const string &s2)
{
    return s1.size() < s2.size();
}
```

See Note

Array and Function

- We very often want to a function to process the data in an array.
- In those cases, array is a function parameter.
- Array parameter is a very special case in C++. The array name ALWAYS be followed by **an empty bracket**.

```
void set_data(int numbs[], int size);

void get_data(const int numbs[], int size);
```

- The effect practically looks like pass-by-reference.

See Note

Functions: Return

- Every return in a function with a return type other than void must return a value.
- Return a nonreference type
  - Value returned by a function initializes a temporary (object) created at the point when the call was made.
  - Return value is copied into the temporary at the calling site
- Return a reference type
  - When a function returns a reference type, the return value is not copied. Instead, the object itself is returned.
- See note.

```
// Disaster: Function returns a reference to a local object
const string &manip(const string& s)
{
    string ret = s;
    // transform ret in some way
    return ret; // Wrong: Returning reference to a local object!
}
```

-- This function will fail at run time because it returns a reference to a local object.
-- When the function ends, the storage in which ret resides is freed. The return value refers to memory that is no longer available to the program.

Never Return a Reference to a Local Object!

(EFC++ Item 23: Don't try to return a reference when you must return an object)

I expect you have learned or can self-learn …

6.3.2 Recursive function

6.4 Overloaded functions

6.5.1 Default arguments

Until Next Time

- Lab starts at 6:00 pm on Thurs.
- HW2 will be due at 0900 pm on 10.02
- [Reading] Chapter 7.